

Real-time Software Telemetry Processing System (RT-STPS) User's Guide

Version 5.6

May 2014



GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

Table of Contents

1	General	1
2	Software Description	1
3	Software Version	2
4	Prerequisites	3
5	Program Inputs and Outputs	3
6	Installation and Configuration	3
6.1	Linux Platform Installation	3
6.1.1	Requirements	4
6.1.2	leapsec File Configuration	4
6.1.3	Installation	4
6.1.4	Configure RT-STPS	5
6.1.5	Package Layout	6
6.1.6	Testing the Installation	6
6.1.6.1	Start the Server Manually	6
6.1.6.2	Execute the Viewer	7
6.1.6.3	Execute the Sender	8
6.1.6.4	Inspect the Results	8
6.1.6.5	Stop the Server	9
6.1.7	Creating Launchers	9
6.1.8	Firewall Configuration	9
6.2	Windows Platform Installation	9
6.2.1	Requirements	10
6.2.2	leapsec File Configuration	10
6.2.3	Installation	10
6.2.4	Configure RT-STPS	11
6.2.5	Package Layout	11
6.2.6	Testing the Installation	12
6.2.6.1	Start the Server Manually	12
6.2.6.2	Execute the Viewer	12
6.2.6.3	Execute the Sender	13
6.2.6.4	Inspect the Results	14
6.2.6.5	Stop the Server	15
6.2.7	Creating Shortcuts	15
6.2.8	Firewall Configuration	15
6.3	Java Setup and Performance Notes	15
7	Program Operation	15
7.1	Starting the Server	16
7.1.1	Command Line Script	16
7.1.2	Java Service Wrapper (Linux Only)	16
7.2	Viewer	17
7.2.1	Viewer Function Summary	17
7.2.1.1	Menu Bar	17
7.2.1.2	File Menu	17

7.2.1.3	Commands Menu.....	17
7.2.1.4	Status Menu.....	18
7.2.1.5	Button Bar	18
7.3	Sender	18
7.4	Batch Mode	18
7.5	Logging	19
7.6	Stopping the Server	19
7.6.1	Command Line Script	19
7.6.2	Java Service Wrapper (Linux Only)	20
7.7	Rebuilding.....	20
8	Raw Data Record (RDR) Creation	21
8.1	Supported RDRs	21
8.2	Test Data	21
8.3	Suomi NPP Configuration File for RDRs.....	21
8.4	Processing the Test Data.....	22
8.5	RDR Processing Status	22
8.6	Expected Output	22
8.7	Reference Sources	22
9	Additional Command Line Tools	23
9.1	Getstatus.....	23
9.2	Load	23
9.3	Shutdown	23
9.4	Version.....	24
9.5	Rate Buffering Program	24
10	Additional Server Configuration Options	26
10.1	Configuring the Log.....	27
10.2	Automatic Setup.....	27
10.3	Adding a Second Server	27
10.4	Provided Configuration Files	28
11	Alternate Server Interface	28
12	Understanding the Configuration Files	29
12.1	Frame Synchronizer Element	29
12.2	Cyclic Redundancy Check (CRC) Decoder Element	32
12.3	Reed-Solomon Decoder Element	33
12.4	Spacecrafts.....	34
12.5	Terra Decoder Element.....	34
12.6	CADU Service Element.....	35
12.7	CCSDS Services Element	35
12.7.1	VCDU Service Element	35
12.7.2	Bitstream Service Element	36
12.7.3	Path Service Element.....	37
12.8	Packets Element.....	39
12.9	Output Channels Element.....	41
12.9.1	File Output Channel Element	41
12.9.2	Annotation	42
12.9.3	Socket Output Channel Element	43

12.9.3.1 Sorcerer Output Channel Element	45
12.9.3.2 Application ID Sub-element.....	46
12.9.3.3 Packet Length Sub-element.....	47
12.9.4 RDR Output Element.....	47
12.9.4.1 Packet List	48
12.9.4.2 Supported Application Identifiers	48
12.10 Links Element	49
Table 1. frame_sync Element.....	30
Table 2. crc Element	32
Table 3. reed_solomon Element	33
Table 4. spacecrafts Element.....	34
Table 5. cadu_service Element	35
Table 6. vcd� Element.....	36
Table 7. bitstream Element	37
Table 8. path Element	38
Table 9. pklink Element.....	39
Table 10. packets Element.....	40
Table 11. file Element.....	41
Table 12. Packet Annotation	42
Table 13. Frame Annotation.....	42
Table 14. socket Element.....	44
Table 15. sorcerer Element	45
Table 16. appidSub-element	46
Table 17. packetLengthSub-element.....	47
Table 18. RDR Element	48
Table 19. Supported Application Identifiers.....	48
Table 20. Supported Application Identifiers by RDR	49
Table 21. links Element.....	49
Table 22. Predefined Labels	49
Figure 1. RT-STPS Architecture.....	2
Figure 2. RT-STPS Viewer	7
Figure 3. RT-STPS Sender	8
Figure 4. RT-STPS Viewer	13
Figure 5. RT-STPS Sender	14
Figure 6. RDR Creation Process.....	21

1 General

The NASA Goddard Space Flight Center's (GSFC) Direct Readout Laboratory (DRL), Code 606.3 developed the Real-time Software Telemetry Processing System (RT-STPS) software for the International Polar Orbiter Processing Package (IPOPP).

Users must agree to all terms and conditions in the Software Usage Agreement on the DRL Web Portal before downloading this software.

Software and documentation published on the DRL Web Portal may occasionally be updated or modified. The most current versions of DRL software are available at the DRL Web Portal:

<http://directreadout.sci.gsfc.nasa.gov>

Questions relating to the contents or status of this software and its documentation should be addressed to the DRL via the Contact DRL mechanism at the DRL Web Portal:

<http://directreadout.sci.gsfc.nasa.gov/?id=dspContent&cid=66>

2 Software Description

RT-STPS ingests raw telemetry data and produces products, including sorted Consultative Committee for Space Data Systems (CCSDS) packets and Virtual Channel Data Units (VCDUs). RT-STPS functions in two modes: Standalone, or as an IPOPP plug-in (Linux only), and it supports a variety of output formats.

Installed as a server RT-STPS will operate continuously, receiving data from a port or a file and outputting results to files and sockets as specified in a configuration file. A separate interface can be used to invoke RT-STPS from the command line.

The RT-STPS package includes two main utilities: the viewer and the sender. The viewer displays the progress of the server as it runs, and it can be used to load server configuration files. The sender copies a raw data file to the server for processing.

The RT-STPS architecture is depicted in Figure 1. Each server component performs a typical part of the overall CCSDS processing from raw telemetry frames to packets.

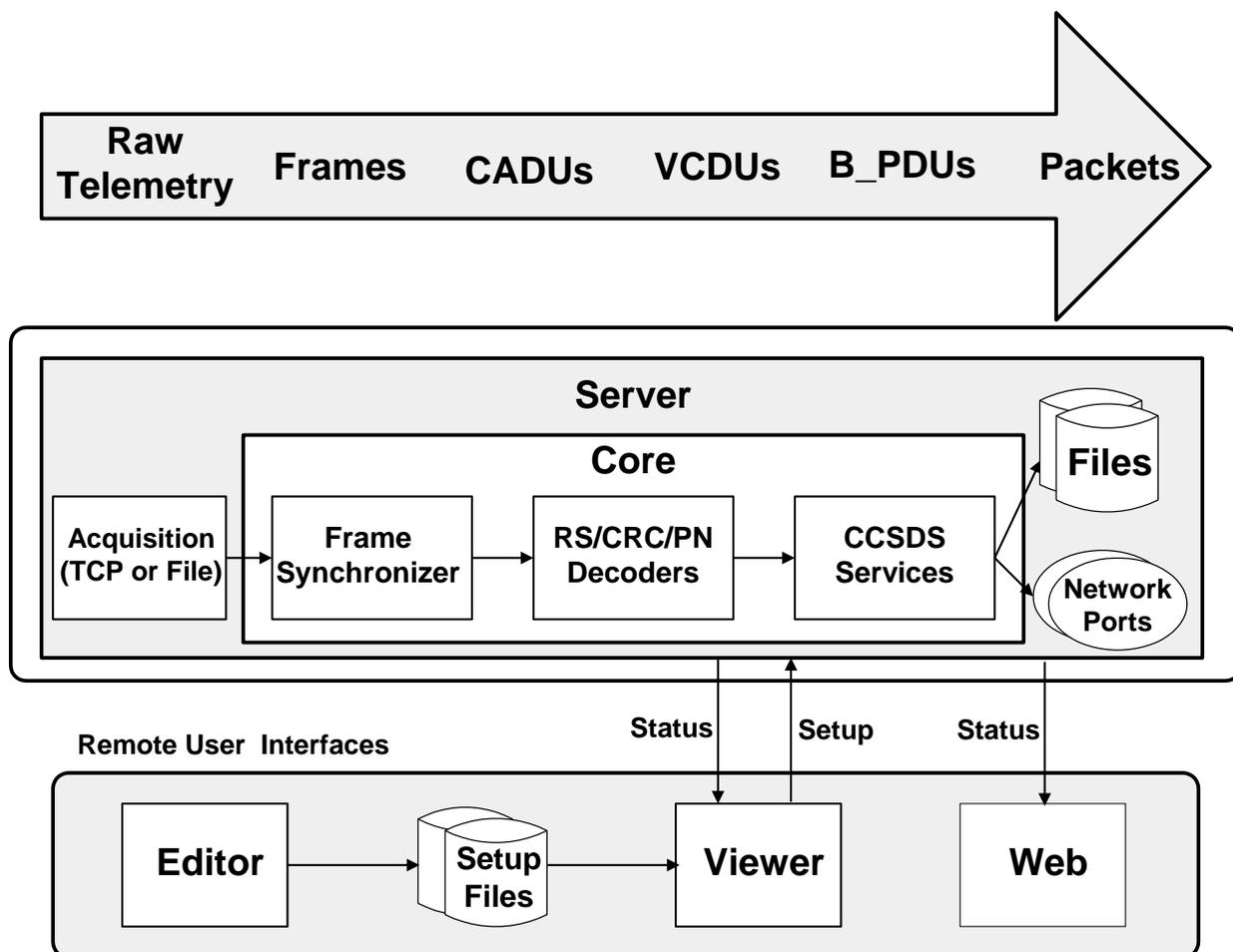


Figure 1. RT-STPS Architecture

3 Software Version

This software package contains RT-STPS Version 5.6. Copyright 1999-2007, United States Government as represented by the Administrator for the National Aeronautics and Space Administration. All Rights Reserved.

Enhancements to Version 5.6 include:

- improved HDF5 resource management;
- updated Linux Java Service Wrapper (JSW) when RT-STPS is implemented as a system service;
- improved Reed-Solomon capability for interleave depths 1, 2, 3, and 5;
- improved packaging for multi-platform use.

4 Prerequisites

To run this package, you must have Java Development Kit (JDK) (Java 1.6.0_25 or higher) installed on your computer. The JDK is required by some scripts which use the server version of the Java Virtual Machine (JVM), and it is necessary if the RT-STPS distribution will be rebuilt. The bin directory of the JDK must be added to the beginning of the PATH environment variable. Placing the bin directory at the beginning of the PATH environment variable ensures use of the correct Java version rather than the system default.

5 Program Inputs and Outputs

RT-STPS ingests raw CCSDS-compliant frames that may be Pseudo-noise (PN)-encoded or Reed-Solomon (RS)-encoded and outputs VCDUs or packets into the following formats:

- a) RDR files: SNPP Visible Infrared Imaging Radiometer Suite (VIIRS), Advanced Technology Microwave Sounder (ATMS), Cross-track Infrared Sounder (CrIS), and Ozone Mapping Profiler Suite (OMPS);
- b) Production Data Set (PDS) (packet file and Construction Record [CSR]) file pairs;
- c) File: header, trailer, and no annotation;
- d) Sockets.

6 Installation and Configuration

RT-STPS can be installed on Linux or Microsoft Windows platforms. Installation and configuration instructions for each platform are provided in separate sections: for Linux refer to section 6.1, and for Windows refer to section 6.2. Program operation instructions for both Linux and Windows installations are contained in section 7.

The RT-STPS_5.6_testdata.tar.gz (.zip for Windows) file contains a sample Suomi NPP raw input data file ("rt-stps_npp_testdata.dat") that may be processed with RT-STPS to test RT-STPS following installation and configuration. Instructions are contained in section 6.1.6 (Linux) and section 6.2.6 (Windows). The "rt-stps_npp_testdata.dat" file will produce RDRs for the Suomi NPP VIIRS, ATMS, CrIS, and OMPS instruments. Refer to section 8 for additional information.

6.1 Linux Platform Installation

RT-STPS may be installed in two modes of operation on Linux: Standalone or as an IPOPP plug-in. In either case, RT-STPS should be installed into a 'dr!' directory such as '/home/dr!'.

When installing RT-STPS on a computer where IPOPP has been installed previously, be sure to install RT-STPS in the existing 'dr!' directory so it may configure itself properly. Before doing so copy any customized configuration files from the previous installation to a safe location so they will not accidentally be overwritten during installation.

If you are upgrading from a previous version of RT-STPS, first stop the RT-STPS services, then delete the existing 'rt-stps' directory.

6.1.1 Requirements

RT-STPS requires the Sun/Oracle JDK Java 1.6.0_25 or higher. Download and install the JDK for Linux according to the Oracle/Sun instructions.

The RT-STPS package comes pre-compiled using a Java 1.6 64-bit JDK. A 64-bit version of Linux is required for proper operation.

The bin directory of the JDK must be added to the beginning of the PATH environment variable. Placing the bin directory at the beginning of the PATH environment variable ensures use of the correct Java version rather than the system default. Refer to section 6.3 for more details regarding Java requirements.

RT-STPS has been tested with the following Linux distributions:

- a) Fedora 18 X86_64;
- b) CentOS Linux 6.4 X86_64;
- c) OpenSUSE Linux 12.1 X86_64;
- d) Kubuntu 13.04 X86_64

6.1.2 leapsec File Configuration

RT-STPS requires a leapsec file to calculate the current number of leap seconds to use in time calculations. An up-to-date leapsec file is required for RT-STPS operation. A leapsec file is included in the RT-STPS software package. For correct time calculations, please ensure that an up-to-date leapsec.dat file is located in the root directory of RT-STPS. Leapsec files are available at:

<ftp://is.sci.gsfc.nasa.gov/ancillary/temporal/>

The leapsec filename must follow one of the two following formats:

- a) leapsec.dat
- b) leapsec.YYYYMMDDNN.dat, e.g. leapsec.2013042201.dat

6.1.3 Installation

NOTE: These installation instructions assume that the RT-STPS package will be installed in '/home/drl'.

If the RT-STPS is currently running, stop it by executing the following from the command line:

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./jsw/bin/rt-stps-server.sh stop
```

If installing RT-STPS on a computer where RT-STPS has been installed previously, first save any customized configuration files to another location before removing the existing 'rt-stps' directory.

Then to install the new package:

- 1) create a user account (if it does not already exist) under which the server will run;
- 2) make a 'drl' subdirectory in which the RT-STPS will be installed;
- 3) copy the downloaded RT-STPS_5.6.tar.gz file to the 'drl' directory;
- 4) decompress RT-STPS_5.6.tar.gz using the command:

```
$ tar xzvf RT-STPS_5.6.tar.gz
```

An 'rt-stps' directory containing the contents of the RT-STPS package should now be installed in the 'drl' directory. The RT-STPS user must own and have full read/write permissions to the 'drl' and 'rt-stps' directories. Copy any saved configuration files to the new package location as is appropriate for your system.

Copy the downloaded RT-STPS_5.6_testdata.tar.gz file to the 'rt-stps' directory. Decompress and un-archive the RT-STPS_5.6_testdata.tar.gz file:

```
$ tar -xzf RT-STPS_5.6_testdata.tar.gz
```

6.1.4 Configure RT-STPS

Change to the 'drl/rt-stps' directory and run the installation script:

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./install.sh
```

In order to make the script executable, it may be necessary to run the command:

```
chmod +x install.sh
```

The script looks for a previous installation of IPOPP; if it finds one, it will assume that RT-STPS is being installed in IPOPP Mode and the following message will appear:

Configuring RT-STPS for IPOPP Mode.

If IPOPP software has not been installed previously, then the user will receive the following message:

Configuring RT-STPS for Standalone Mode.

In either case when configuration is complete, the user will receive the following message:

Configuration complete.

This message indicates that an RT-STPS package has been installed and configured successfully.

6.1.5 Package Layout

Once the package has been unpacked and installed, the 'rt-stps' directory should contain the following contents:

- bin/ – command line applications
- classes/ – compiled java classes directory
- config/ – XML configuration files
- data/ – telemetry data input
- docs/ – javadoc target directory
- images/ – various icon and graphic images
- jsw/ – Java Service Wrapper software for Linux
- lib/ – jar files including rt-stps.jar and HDF libraries
- src/ – source tree
- testdata/ – contains the "rt-stps_npp_testdata.dat" file and test output products
- build.sh – Linux build script
- build_javadoc.sh – Linux build javadoc script
- install.sh – installation script for Linux
- leapsec.dat – leapsec file used for accurate time calculations
- rt-stps.dtd – the Data Type Definition (DTD) for the XML configuration files
- rt-stps.policy – policy statement
- SUA Open Source IPOPP GSC-15570-1.pdf – software use agreement
- VERSIONLOG – version info

6.1.6 Testing the Installation

The server places its home directory as the 'rt-stps' directory, and the configuration files specify that output will go to the './data' directory. RT-STPS places output into the 'data' directory at the same level as the 'rt-stps' directory (i.e., 'rt-stps/./data').

After testing the installation the user may wish to edit the configuration files to specify a preferred destination directory. Refer to section 12.9.

6.1.6.1 Start the Server Manually

To test the installation the server must be started from the command line. Ensure that you are in the 'rt-stps' directory and then start the server by issuing the start command to the JSW (Java Service Wrapper):

```
/home> cd /home/drl/rt-stps
```

```
/home/drl/rt-stps> ./jsw/bin/rt-stps-server.sh start
```

The JSW captures all output messages to a log file in 'rt-stps/jsw/logs'. The overall status of the server can also be checked by issuing the status command to the JSW:

```
/home/drl/rt-stps> ./jsw/bin/rt-stps-server.sh status
```

This command will return its current process identifier and a message confirming it is running.

6.1.6.2 Execute the Viewer

Next start the viewer application. The viewer allows the user to load configuration files and to view processing status. From the command line in the 'rt-stps' directory enter:

```
/home/drl/rt-stps> ./bin/viewer.sh &
```

This command will bring up the viewer Graphical User Interface (GUI), depicted in Figure 2. Use the Load button to load the "npp_with_omps.xml" configuration file from the 'rt-stps/config' directory. Use the file dialog to find this directory and file. Once the file loads, click on the Go button.

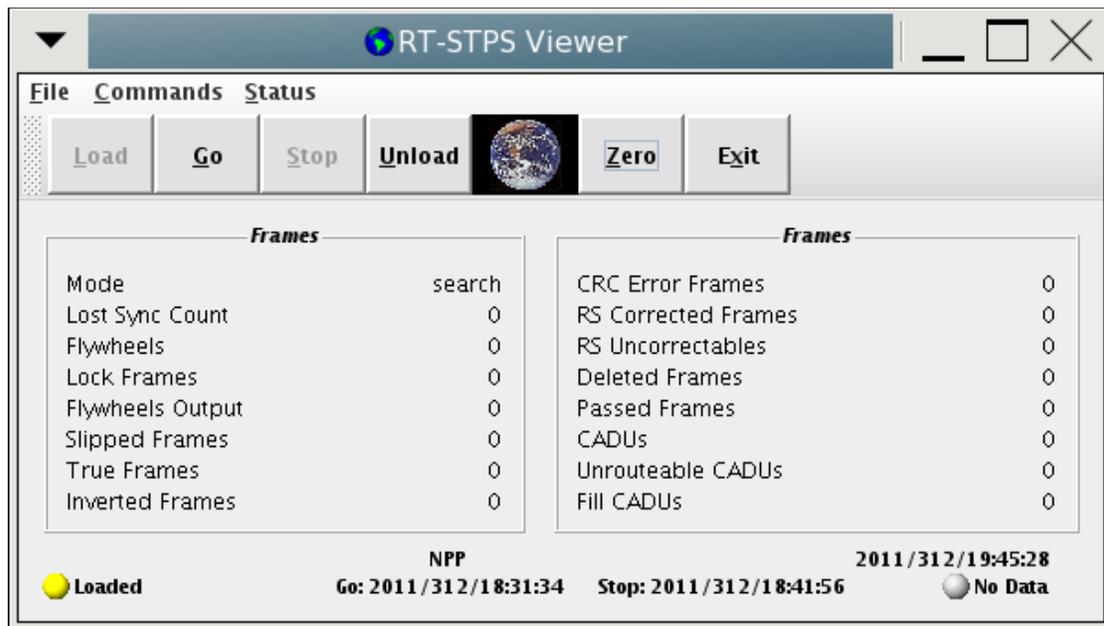


Figure 2. RT-STPS Viewer

6.1.6.3 Execute the Sender

Next start the sender application. The sender allows the user to send telemetry data files to the RT-STPS server for processing. It also displays the percentage of the file that has been sent to the server. From the command line in the 'rt-stps' directory, enter:

```
/home/drl/rt-stps> ./bin/sender.sh &
```

This will bring up the sender GUI and place the command into the background.

Click on the File button. Use the File dialog to select the "rt-stps_npp_testdata.dat" file from the 'testdata/input' directory extracted from the RT-STPS_5.6_testdata.tar.gz file, and click on the Go button to send the file to the server for processing. The sender is depicted in Figure 3.

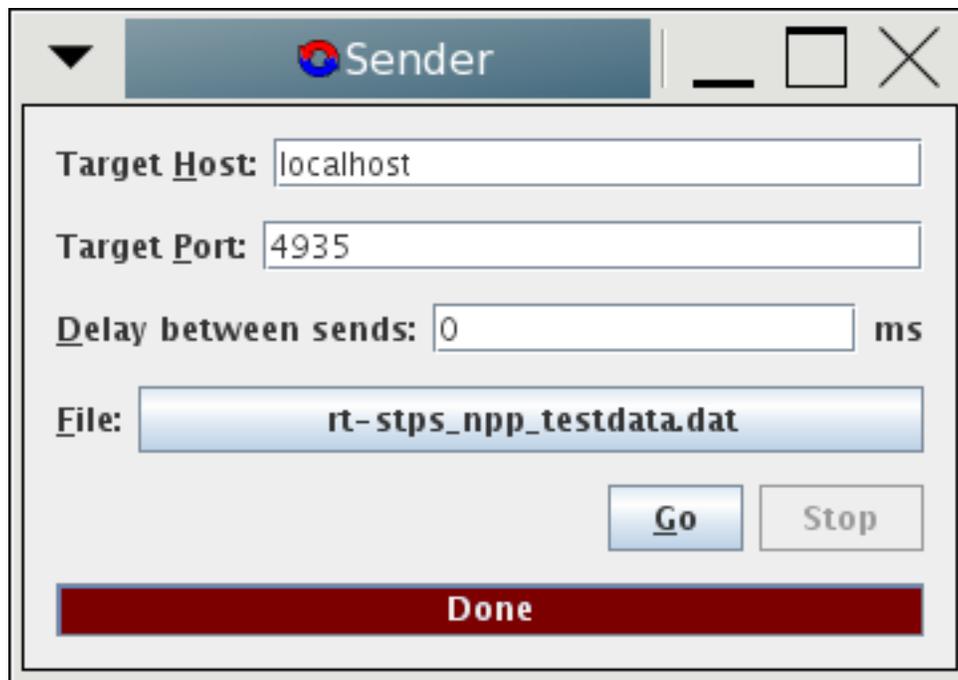


Figure 3. RT-STPS Sender

6.1.6.4 Inspect the Results

Once processing is complete (the viewer status buttons will show that no more data are being processed, and the sender will show that it is done), list and inspect the contents of the 'rt-stps/./data' directory. It should contain RDR files with the following filenames (the dates of the creation fields that start with a 'c' will be different):

```
RATMS-RNSCA_npp_d20120529_t1657471_e1702031_b00001_c20140502204548171000_all-_dev.h5
```

```
RCRIS-RNSCA_npp_d20120529_t1657471_e1702031_b00001_c20140502204547471000_all-_dev.h5
```

```
RNSCA-ROLPS_npp_d20120529_t1657344_e1702338_b00001_c20140502204548284000_all-_dev.h5
```

RNSCA-RONPS_npp_d20120529_t1657424_e1701268_b00001_c20140502204548311000_all-_dev.h5

RNSCA-ROTCS_npp_d20120529_t1657424_e1702042_b00001_c20140502204548297000_all-_dev.h5

RNSCA-RVIRS_npp_d20120529_t1656471_e1702285_b00001_c20140502204546826000_all-_dev.h5

Test output products are available in the 'rt-stps/testdata/output' directory. The test output products serve as an indicator of expected program output. Use a comparison utility (such as diff, h5diff, etc.) to compare your output products to those provided in the 'rt-stps/testdata/output' directory. Locally generated products may differ slightly from the provided test output products because of differences in machine architecture or operating systems.

6.1.6.5 Stop the Server

Once the test is complete the server may be stopped by issuing the stop command to the JSW:

```
/home/drl/rt-stps> ./jsw/bin/rt-stps-server.sh stop
```

WARNING: Since the viewer/sender requires the RT-STPS server to be running, it is advised that the sender and viewer be closed prior to stopping the server.

6.1.7 Creating Launchers

Desktop launchers to run the viewer and sender may be created to start these applications. The creation of launchers varies between the different types of Linux distributions and their desktop environments. In all cases the commands described in section 6.1.6.2 and section 6.1.6.3 will apply.

6.1.8 Firewall Configuration

A firewall may in some cases prevent RT-STPS from running correctly, and it may need to be disabled or configured to allow access for certain ports. The server accepts data by default on port 4935. The output port numbers, if any, are defined in the configuration files. The viewer initially connects through port 1099; afterwards, the viewer and server communicate through anonymous ports.

6.2 Windows Platform Installation

RT-STPS will be installed in Standalone mode on Windows.

When installing RT-STPS on a computer where RT-STPS has been installed previously, copy any customized configuration files from the previous installation to a safe location so they will not accidentally be overwritten during installation.

If you are upgrading from a previous version of RT-STPS, first stop the RT-STPS services, then delete the existing 'rt-stps' directory.

NOTE: This version of RT-STPS for Windows does not support the JSW (Java Service Wrapper).

6.2.1 Requirements

RT-STPS requires the Sun/Oracle JDK Java 1.6.0_25 or higher. Download and install the JDK for Windows according to the Oracle/Sun instructions. The RT-STPS package is pre-compiled using the Java 1.6.0_25 64-bit JDK with HDF 64-bit.

The 'bin' directory of the JDK must be added to the beginning of the PATH environment variable so that the proper version Java executable is used. The easiest way to set the PATH is to use the System Properties dialogue. Refer to section 6.3 for more details regarding Java requirements.

RT-STPS has been tested with Microsoft Windows 7 64-bit.

6.2.2 leapsec File Configuration

RT-STPS requires a leapsec file to calculate the current number of leap seconds to use in time calculations. An up-to-date leapsec file is required for RT-STPS operation. A leapsec file is included in the RT-STPS software package. For correct time calculations, please ensure that an up-to-date leapsec.dat file is located in the root directory of RT-STPS. Leapsec files are available at:

<ftp://is.sci.gsfc.nasa.gov/ancillary/temporal/>

The leapsec filename must follow one of the two following formats:

- a) leapsec.dat
- b) leapsec.YYYYMMDDNN.dat, e.g. leapsec.2013042201.dat

6.2.3 Installation

NOTE: These installation instructions assume that the RT-STPS package has been installed in 'C:\Users\drl'.

If RT-STPS is currently running, stop it by executing the command:

```
C:\Users> cd C:\Users\drl\rt-stps\bin
C:\Users\drl\rt-stps\bin> stop.bat
```

If installing RT-STPS on a system where RT-STPS has been installed previously, first save any customized configuration files to another location before removing the existing 'rt-stps' directory.

Then to install the new package:

- 1) create a user account (if it does not already exist) under which the server will run;

- 2) make a 'drl' subdirectory in which the RT-STPS will be installed;
- 3) copy the downloaded RT-STPS_5.6.zip file to the 'drl' directory;
- 4) decompress the RT-STPS_5.6.zip file directly into the 'drl' directory.

An 'rt-stps' directory containing the contents of the RT-STPS package should now be installed in the 'drl' directory. The RT-STPS user must own and have full read/write permissions to the 'drl' and 'rt-stps' directories. Copy any saved configuration files to the new package location as is appropriate for your system.

Copy the downloaded RT-STPS_5.6_testdata.zip file to the 'drl' directory. Decompress and un-archive the RT-STPS_5.6_testdata.zip file directly into the 'drl' directory; if asked to merge directory contents with the existing 'rt-stps' directory, select "Yes".

6.2.4 Configure RT-STPS

Change to the 'drl\rt-stps' directory and run the installation script:

```
C:\Users\> cd C:\Users\drl\rt-stps
C:\Users\drl\rt-stps> install.bat
```

The user will receive the following message:

Configuring RT-STPS for Standalone Mode.

When configuration is complete, the user will receive the following message:

Configuration complete.

These messages indicate that an RT-STPS package has been installed and configured successfully.

6.2.5 Package Layout

Once the package has been unpacked and installed, the 'rt-stps' directory should contain the following contents:

- bin\ – command line applications
- classes\ – compiled java classes directory
- config\ – XML configuration files
- data\ – telemetry data input
- docs\ – javadoc target directory
- images\ – various icon and graphic images
- lib\ – jar files including rt-stps.jar and HDF libraries
- logs\ – location of RT-STPS server log files
- src\ – source tree
- testdata\ – contains the "rt-stps_npp_testdata.dat" file and test output products

- build.bat – Windows build script
- build_javadoc.bat – Windows build javadoc script
- install.bat – installation script for Windows
- leapsec.dat – leapsec file used for accurate time calculations
- rt-stps.dtd – the Data Type Definition (DTD) for the XML configuration files
- rt-stps.policy – policy statement
- SUA Open Source IPOPP GSC-15570-1.pdf – software use agreement
- VERSIONLOG – version info

6.2.6 Testing the Installation

By default the server uses 'rt-stps\..\data' directory to store processed results. This location is specified in the configuration files in the 'rt-stps\config' directory. After testing the installation the user may wish to edit the configuration files to change the destination directory from 'rt-stps\..\data' to a preferred location. Refer to section 12.9.

6.2.6.1 Start the Server Manually

The JSW is not supported under Windows, so it's necessary to start the server from the command prompt using the provided server script in the 'rt-stps\bin' directory. From the command line in the 'rt-stps' directory, enter:

```
C:\Users> cd C:\Users\dr\rt-stps
C:\Users\dr\rt-stps> bin\server.bat
```

The server captures all output to a log file named "rt-stps-server.log" in the 'rt-stps\logs' directory. The "rt-stps-server.log" file will appear in the 'rt-stps\logs' directory after a configuration file has been loaded with the viewer, as described in section 6.2.6.2.

To check that the server is running, use the Windows Task Manager window to find the javaw associated with the server.

6.2.6.2 Execute the Viewer

Next start the viewer application from the command prompt. The viewer allows the user to load configuration files and view processing status. From the command line in the 'rt-stps' directory, enter:

```
C:\Users\dr\rt-stps> bin\viewer.bat
```

This command will bring up the viewer Graphical User Interface (GUI), depicted in Figure 4. Use the Load button to load the "npp_with_omps.xml" configuration file from the 'rt-stps\config' directory. Use the file dialog to find this directory and file. Once the file loads, select the Go button. The viewer GUI is depicted in Figure 4.

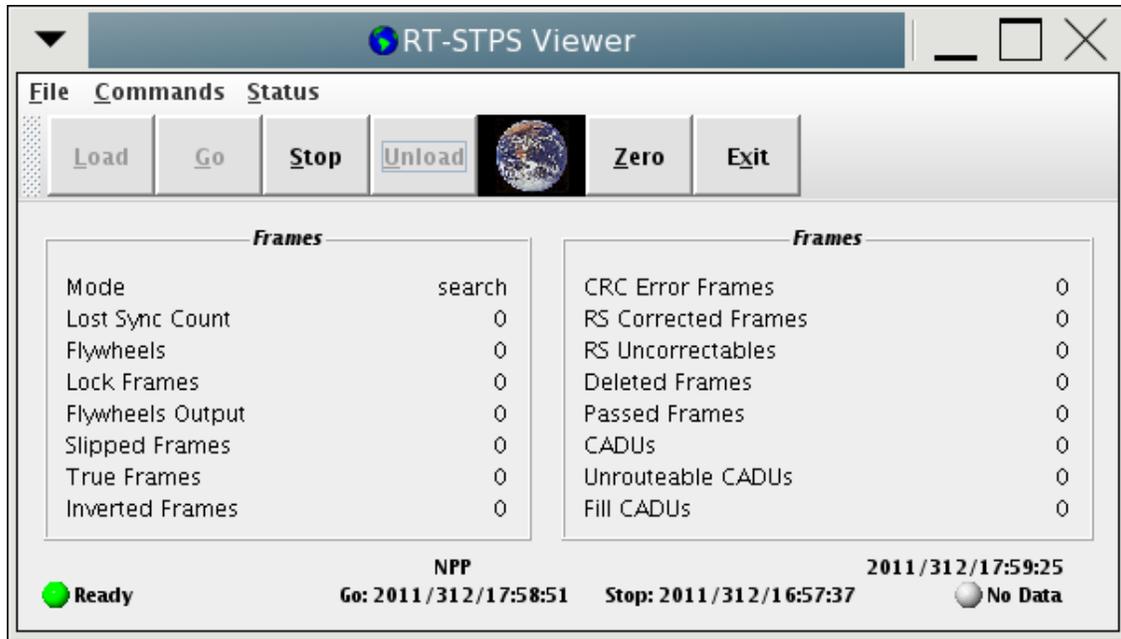


Figure 4. RT-STPS Viewer

6.2.6.3 Execute the Sender

Next start the sender application. The sender allows the user to send telemetry data files to the server for processing. It also displays the percentage of the file that has been sent to the server. From the command line in the 'rt-stps' directory, enter:

```
C:\Users\dr\rt-stps> bin\sender.bat
```

This will bring up the sender GUI. On the sender GUI select the File button and use the file dialogue to find and select the "rt-stps_npp_testdata.dat" file in the 'testdata\input' directory extracted from the RT-STPS_5.6_testdata.zip file. Once the file is selected, click on the Go button to send the file to the server for processing. The sender is depicted in Figure 5.

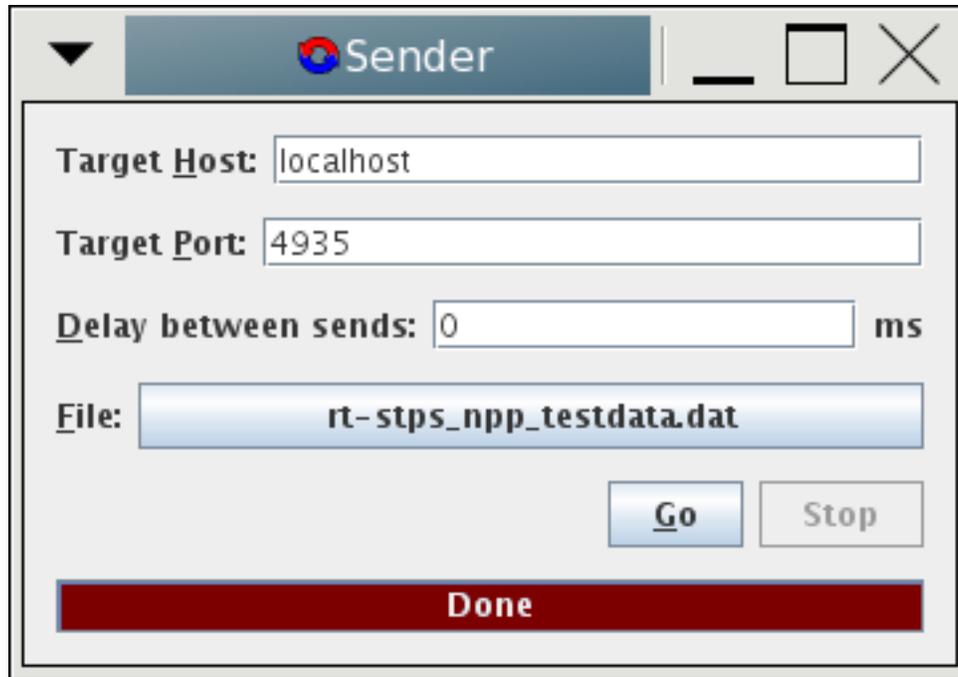


Figure 5. RT-STPS Sender

6.2.6.4 Inspect the Results

Once processing is complete (the viewer status buttons will show that no more data is being processed, and the sender will show that it is Done), inspect the contents of the 'rt-stps\.\data' directory. It should contain RDR files with the following filenames (the date of the creation fields that start with a 'c' will be different):

```
RATMS-RNSCA_npp_d20120529_t1657471_e1702031_b00001_c20140502204548171000_all-_dev.h5  
RCRIS-RNSCA_npp_d20120529_t1657471_e1702031_b00001_c20140502204547471000_all-_dev.h5  
RNSCA-ROLPS_npp_d20120529_t1657344_e1702338_b00001_c20140502204548284000_all-_dev.h5  
RNSCA-RONPS_npp_d20120529_t1657424_e1701268_b00001_c20140502204548311000_all-_dev.h5  
RNSCA-ROTCS_npp_d20120529_t1657424_e1702042_b00001_c20140502204548297000_all-_dev.h5  
RNSCA-RVIRS_npp_d20120529_t1656471_e1702285_b00001_c20140502204546826000_all-_dev.h5
```

Test output products are available in the 'rt-stps\testdata\output' directory. The test output products serve as an indicator of expected program output. Use a comparison utility (such as h5diff, etc.) to compare your output products to those provided in the 'rt-stps\testdata\output' directory. Locally generated products may differ slightly from the provided test output products because of differences in machine architecture or operating systems.

6.2.6.5 Stop the Server

Once the test is complete the server may be stopped by running the stop script provided in the 'rt-stps\bin' directory. From the command line in the 'rt-stps' directory, enter:

```
C:\Users\dr\rt-stps> bin\stop.bat
```

WARNING: Since the RT-STPS viewer/sender requires the RT-STPS server to be running, it is advised that the sender and viewer be stopped prior to stopping the server.

6.2.7 Creating Shortcuts

Create shortcuts to the viewer and sender as follows:

- 1) In the 'rt-stps\bin' directory, right-click the "viewer.bat" and "sender.bat" files and select "Create Shortcut".
- 2) Move the shortcuts to the desired location (e.g., the Desktop).
- 3) Rename the shortcuts "RT-STPS Viewer" and "RT-STPS Sender" respectively.
- 4) Right-click the shortcuts and select "Properties". Modify the "Start in:" field to be the absolute path of the 'rt-stps' directory. Change the "Run:" selection to "Minimized".

6.2.8 Firewall Configuration

A firewall may in some cases prevent RT-STPS from running correctly, and it may need to be disabled or configured to allow access for certain ports. The server accepts data by default on port 4935. The output port numbers, if any, are defined in the configuration files. The viewer initially connects through port 1099; afterwards, the viewer and server communicate through anonymous ports.

6.3 Java Setup and Performance Notes

Many of the scripts (batch files) in the RT-STPS 'bin' directory employ the "-server" option to increase performance, as the Sun Java Virtual Machine (JVM) server version may be somewhat faster than the client version of the JVM. By default the JRE comes with the client JVM only, and to use the server JVM, you must install the JDK. If the JDK is installed ensure that is on the PATH before the JRE.

If the "-server" option is used in a script and the server JVM is not properly installed, Windows will issue an error message as follows:

```
Error: no 'server' JVM at "...\jreX.X.X\bin\server\jvm.dll"
```

7 Program Operation

This section includes instructions for both Linux and Windows platforms.

RT-STPS can be started as a server from a command line server script, or from the

JSW (Linux only). A configuration file is required to describe the telemetry format specifications and the data outputs for the spacecraft of interest. Typically, the setup configuration file will be loaded prior to each pass. Sample setup files are stored in the 'rt-stps/config' directory.

The 'rt-stps/./data' directory is the default target for output file results for both the Linux and Windows version of RT-STPS.

A sample raw telemetry file named "rt-stps_npp_testdata.dat" resides in the 'testdata/input' directory extracted from the RT-STPS_5.6_testdata.tar.gz (.zip) file. It contains packet data from the VIIRS, ATMS, CrIS, and OMPS instruments aboard the Suomi NPP satellite. A corresponding "npp_with_omps.xml" configuration file resides in the 'config' directory. Use it to produce VIIRS, ATMS, CrIS, and OMPS RDRs. Instructions for testing RT-STPS using these sample files are contained in sections 6.1.6 (Linux) and 6.2.6 (Windows). The RT-STPS package also includes the "aqua.xml" and "terra.xml" configuration files.

7.1 Starting the Server

The server may be started from a command line server script, or from the JSW (Linux only). For Linux, it is strongly recommended to start the server from the JSW.

7.1.1 Command Line Script

Change to the 'rt-stps' directory and run the server script:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/server.sh &
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\server.bat
```

7.1.2 Java Service Wrapper (Linux Only)

Change to the 'rt-stps' directory and issue the start command to the JSW:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./jsw/bin/rt-stps-server.sh start
```

These commands initiate the server process in the background. When ready, the "Ready to serve" status message is logged to the appropriate location. Refer to section 7.5 for details regarding logging.

7.2 Viewer

The viewer may be run from the command line or as a shortcut from the desktop if this has been configured on your system. To run from the command line, change to the 'rt-stps' directory and execute the viewer script:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/viewer.sh &
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\viewer.bat
```

The viewer GUI will appear on the screen (see Figure 4). Use the viewer to configure the server and examine its status, as well as to load and unload configuration files.

7.2.1 Viewer Function Summary

7.2.1.1 Menu Bar

The Menu Bar contains the File, Commands, and Status pull-down menus.

7.2.1.2 File Menu

The File Menu contains the program Exit item.

7.2.1.3 Commands Menu

The Commands Menu contains the following commands to configure and run the server:

- a) Local Load. Displays a Dialog Box to select and load a configuration file stored on the computer executing the viewer.
- b) Remote Load. Displays a Dialog Box to select and load a configuration file stored on the computer executing the server.
- c) Go. Starts server data processing.
- d) Stop. Halts server data processing.
- e) Unload. Removes the current configuration file from the server.
- f) Zero Status. Resets the statistics display.

7.2.1.4 Status Menu

The Status Menu contains menu items to display Path Service Status, Packet Status, the Virtual Channel Status Table, and the Packet Status Table.

7.2.1.5 Button Bar

The button bar contains buttons linked to the Commands Menu items (see Figure 4).

7.3 Sender

The RT-STPS sender (Figure 3) is used to send a raw data file to the server. The sender may be run from the command line or as a shortcut from the desktop if this has been configured on your system. To run from the command line, changed into the 'rt-stps' directory and execute the sender script:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/sender.sh &
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\sender.bat
```

The Target Host should be "localhost", the Target Port number should be "4935", and the delay between sends should be zero. Click on the File button. Use the File dialogue to select the desired raw data file, and click on the Go button to send the file to the server for processing. The server status may be checked using the RT-STPS viewer. When the sender finishes, the server will automatically halt data processing and unload the current configuration file.

7.4 Batch Mode

A batch command performs server processing as a standalone, one-time program. It takes as arguments a configuration file and an input data file containing telemetry frames. The outputs are specified in the configuration file. At the command line, change to the 'rt-stps' directory and run the batch script:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/batch.sh <path-to-config-file> <path-to-data-file>
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps
```

```
C:\Users\drl\rt-stps> bin\batch.bat <path-to-config-file> <path-to-data-file>
```

NOTE: -D properties are not supported through the scripts.

7.5 Logging

There are two types of RT-STPS output messages: status/event messages, and standard output. RT-STPS logs these messages depending on the type of installation and how the RT-STPS server was started:

IPOPP Mode Installations:

In IPOPP mode, server status/event messages are sent to the NISGS Status/Event Logging System (NSLS) by default. These messages indicate the server's current status, and are generated whenever the server's status changes (e.g., a configuration file is loaded).

Standalone Mode Installations:

In Standalone mode, server status/event messages are logged into the "rt-stps-server.log" file in the 'rt-stps/jsw/logs' (Linux) or 'rt-stps\logs' (Windows) directory by default.

In both modes, standard output messages generated by the RT-STPS server are only logged if the server is started from the JSW (Linux only). Standard output messages are logged in rolling log files named "rt-stps-server.log.*" in the 'rt-stps/jsw/logs' directory, which are automatically managed by the JSW.

Additional log options are possible. See section 10 for more information on server configuration details.

7.6 Stopping the Server

The server may be stopped from a command line script or from the JSW (Linux only), depending on how it was started. Because the sender and viewer require the RT-STPS server to be running, it is advised that the sender and viewer be stopped prior to stopping the server.

7.6.1 Command Line Script

If the server was started using the command line server script, change to the 'rt-stps' directory and run the stop script:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/stop.sh
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\stop.bat
```

7.6.2 Java Service Wrapper (Linux Only)

If the server was started from the JSW, change to the 'rt-stps' directory and issue the stop command to the JSW:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./jsw/bin/rt-stps-server.sh stop
```

7.7 Rebuilding

If it is necessary to rebuild the RT-STPS distribution, stop the server (refer to section 7.6) then run the build scripts:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./build.sh
```

Windows

```
C:\Users> cd c:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> build.bat
```

This command will compile the Java source files and create the "rt-stps.jar" file in the 'lib' directory.

Warning for Linux users: The build.sh script calls the Java Remote Method Invocation Compiler (RMIC) "rmic.exe" program to build the remote method invocation portion of RT-STPS. The GNU version of RMIC cannot be used with the RT-STPS build script. Ensure that the Java rmic.exe program is before the GNU RMIC program in the system PATH, so the build.bat file will execute properly. Alternatively, edit the build.sh script and hard-code the absolute path to Java rmic.exe.

Warning for Windows users: The build.bat script calls the "rmic.exe" program to build the remote method invocation portion of RT-STPS. However, if Cygwin is installed on your system, the distribution may include a program of the same name. The Cygwin version of RMIC cannot be used with the RT-STPS build script. Ensure that the Java rmic.exe program is before the Cygwin RMIC program in the system PATH, so that the build.bat script will execute properly. Alternatively, edit the build.bat script and hard-code the absolute path to Java rmic.exe.

8 Raw Data Record (RDR) Creation

8.1 Supported RDRs

RT-STPS Version 5.6 supports the creation of Raw Data Records (RDRs) for the Suomi NPP VIIRS, ATMS, CrIS, and OMPS instruments. Attitude and ephemeris packets included in the input produce the corresponding Spacecraft Diary with each RDR.

8.2 Test Data

The RT-STPS_5.6_testdata.tar.gz (.zip for Windows) file contains a sample Suomi NPP raw input data file named "rt-stps_npp_testdata.dat".

This file may be processed with RT-STPS to produce RDRs for the Suomi NPP VIIRS, ATMS, CrIS, and OMPS instruments.

8.3 Suomi NPP Configuration File for RDRs

RT-STPS includes an "npp_with_omps.xml" configuration file that defines everything necessary to create the Suomi NPP VIIRS, ATMS, CrIS, and OMPS RDRs, as well as to output VIIRS science packets to a socket interface. The "npp_with_omps.xml" configuration file can be used as the basis to create other RDR configurations of interest (e.g., a configuration file dedicated solely to VIIRS Science RDRs, among others). Refer to Section 12.9.4 for additional configuration file details. The RDR creation process is depicted in Figure 6.

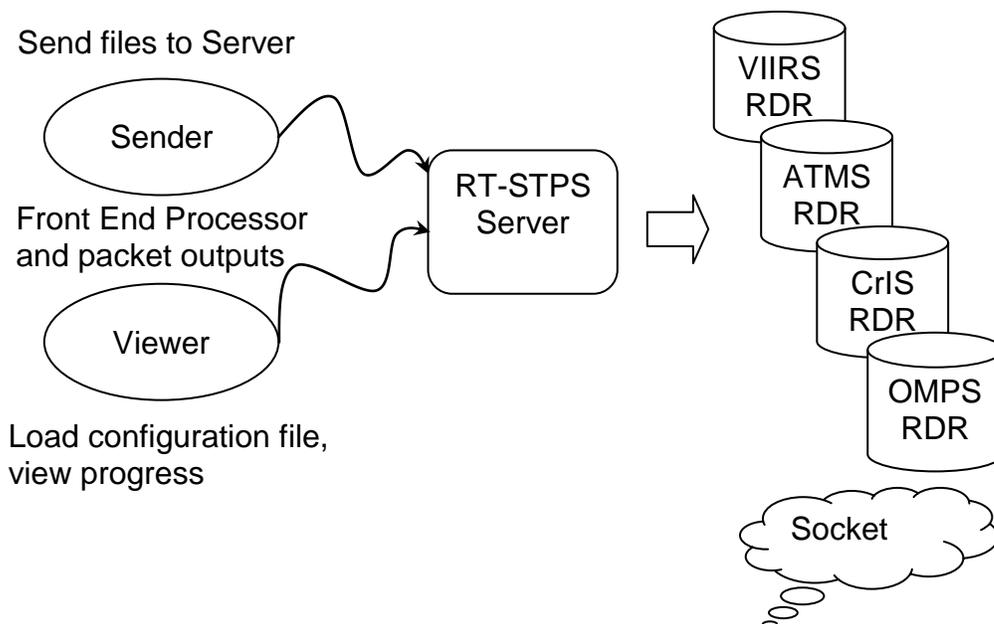


Figure 6. RDR Creation Process

8.4 Processing the Test Data

Decompress and un-archive the RT-STPS_5.6_testdata.tar.gz (.zip) file. Refer to section 6.1.6.1 (Linux) and section 6.2.6.1 (Windows).

Use the viewer to load the configuration file. Refer to section 6.1.6.2 (Linux) and section 6.2.6.2 (Windows) for instructions to execute the viewer. Use the sender to send the 'rt-stps_npp_testdata.dat' file to the server for processing. Refer to section 6.1.6.3 (Linux) and section 6.2.6.3 (Windows) for instructions to execute the sender.

On both Linux and Windows installations, the server uses the 'rt-stps/./data' directory by default to store processed results. Refer to section 6.1.6.4 (Linux) and section 6.2.6.4 (Windows) for additional details.

8.5 RDR Processing Status

When RT-STPS is run in server mode, the RDR processing module updates some status information as it creates granules for the RDR of interest. The status information may be viewed using the RT-STPS viewer, or by using the getstatus script (refer to section 9.1). In batch mode, this information is output to the console directly. Refer to section 7.4 for instructions to run RT-STPS in batch mode.

8.6 Expected Output

Once processing is complete, the viewer status buttons will show that no more data is being processed, and the sender will show that it is done. The 'rt-stps/./data' directory should contain the generated RDR files. Refer to section 6.1.6.4 (Linux) and section 6.2.6.4 (Windows) for more details regarding the expected output when processing the "rt-stps_npp_testdata.dat" raw data file.

8.7 Reference Sources

Details on mission data specifications are contained in the following documents:

- Joint Polar Satellite System (JPSS) Common Data Format Control Book - External Volume II – RDR Formats;
- Joint Polar Satellite System (JPSS) Common Data Format Control Book - External Volume V – Metadata.

Information to identify, distinguish and extract all of the X-band unique source packets from the Suomi NPP mission data streams is contained in the Joint Polar Satellite System (JPSS) Mission Data Format Control Book (MDFCB).

These documents are available via the DRL Web Portal at:

<http://directreadout.sci.gsfc.nasa.gov/?id=dspContent&cid=16>

9 Additional Command Line Tools

Additional scripting tools in the 'rt-stps/bin' directory provide additional functionality to RT-STPS.

9.1 Getstatus

The getstatus script periodically retrieves status information from the active RT-STPS server on the local host until it is stopped. The format is:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/getstatus.sh
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\getstatus.bat
```

9.2 Load

The load script loads a local configuration file into the local server. The format is:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/load.sh <path-to-config-file>
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\load.bat <path-to-config-file>
```

9.3 Shutdown

The shutdown script halts data processing and unloads the current configuration file on the local server. The format is:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/shutdown.sh
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\shutdown.bat
```

9.4 Version

The version script prints RT-STPS version information and exits. The format is:

Linux

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/version.sh
```

Windows

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\version.bat
```

9.5 Rate Buffering Program

RT-STPS includes a rate-buffering program (Rat) that can spool data to a slow target (e.g., such as through a slow network). The server may run slowly because all outputs are processed in one control loop, without internal buffering or internal multi-threading. By using the Rat as an intermediate destination, the server can keep one slow target from slowing the outputs of other targets being serviced at the same time.

Rat may be run on the same computer as the server, or from a remote location. It requires three arguments when it is invoked:

Linux:

```
/home> cd /home/drl/rt-stps  
/home/drl/rt-stps> ./bin/rat.sh <inputPort> <targetHost> <targetPort>
```

Windows:

```
C:\Users> cd C:\Users\drl\rt-stps  
C:\Users\drl\rt-stps> bin\rat.bat <inputPort> <targetHost> <targetPort>
```

Rat listens for socket connections on its input port. When a connection is made, it connects to the target on the target port, forming an end-to-end connection. However, it can only service one input connection at a time. Once connections are established, Rat will buffer input data while sending some data to the slower output connection. Once the input connection closes, Rat will continue to send buffered data to the target until it has no more data, at which point it will close the target connection. Rat forms end-to-end connections after the RT-STPS server has connected to it and will run continuously, listening for and accepting connections until it is terminated.

To stop the Rat program, find its process ID and use the appropriate tools/commands to terminate it. For example:

Linux

Search for the rat.sh process ID, and terminate it. Use commands such as "ps aux" and "grep" to find the correct process ID:

```
/home/drl/rt-stps> ps aux | grep "rat.sh"  
/home/drl/rt-stps> kill <Process-ID>
```

Windows

Search for the javaw.exe process ID associated with rat.bat, and terminate it. Use a command such as "wmic" to display the full command lines and process IDs of all active javaw.exe processes in order to identify the correct javaw.exe process, if there are multiple instances running:

```
C:\Users\drl\rt-stps> wmic process where name="javaw.exe" get  
ProcessID,CommandLine  
C:\Users\drl\rt-stps> taskkill /F /PID <Process-ID>
```

10 Additional Server Configuration Options

The RT-STPS server is configured using command line arguments and system property definitions, or through the JSW configuration file on Linux. In IPOPP mode, much of the configuration is performed automatically during the installation process.

The server takes a *name* argument when invoking it. The full server name then becomes "RtStpsServices." + *name*. The default full server name is "RtStpsServices.A" if no other name is specified.

NOTE: If RT-STPS is installed in IPOPP mode, the NISGS Status/Event Logging System (NSLS) server configuration is performed automatically during the installation process. This configuration overrides any log definitions below.

The server can be configured using several system properties. Set them when using the -D attribute (e.g., "-Dport=4935"). If not set, default values will be used as described below.

- a) -Dconfig=xmlConfigFileName. A configuration file to be used until overridden by a loaded file. The default name is "default.xml".
- b) -Dport=4935. The port number that the Server reads for telemetry data. The default port is 4935.
- c) -DbufferSizeKb=16. The amount of data to accumulate before processing. The default is 16 kb.
- d) -Dsetup=configurationDirectory. The directory where local configuration files are located. If provided, all files must be within the directory tree. The default is 'rt-stps/config'.
- e) -Dlog.stdout. If specified, log messages are written to the standard output.
- f) -Dlog.file=<file>. If specified, log messages are written to <file>.
- g) -Dlog.server=<host:port:tmpDir>. If specified, log messages are sent to the NISGS Status/Event Logging System (NSLS) at host:port (e.g., localhost:3500) and to the temporary directory tmpDir when the NSLS is unavailable.

These arguments and system properties are set in the "rt-stps/bin/server.sh" (Linux) or "rt-stps\bin\server.bat" (Windows) files, or in the configuration file "rt-stps/jsw/conf/rt-stps-server.conf" in the JSW on Linux.

Additional system properties may also be present but should not be modified. For example:

```
...  
#Java Additional Parameters  
Wrapper.java.additional.9=-Dlog.server:MyNSLSServer:3500:/tmp  
...
```

If the server fails to initialize properly, and errors occur before the log service is found, some error messages may go to console instead.

10.1 Configuring the Log

If none of the "-Dlog.*" properties are specified in the server's arguments, log messages are written to the "rt-stps/rt-stps-server.log" file, and standard output is displayed directly on the console.

Additional log options are available as follows:

- a) Write log events to a remote NSLS server: specify `-Dlog.server:hostname:port:tmpDir`. If the remote server is not available, an "event file" will be created with an ".sjo" extension for each event in the specified tmpDir.
- b) Write log events to a file: specify `-Dlog.file=<path-to-log-file>`, and the file will be created if it does not exist.
- c) Write events to stdout: specify `-Dlog.stdout`, and any event will be written to the standard output.

10.2 Automatic Setup

If the server is running but no configuration file is loaded and data arrives on its incoming data port (port 4935), then the server will use the "default.xml" file as its configuration file.

Similarly, if the server is running but has not been re-configured with a new configuration file, and data unexpectedly arrive on its incoming data port, then the server will use the previous configuration file.

If the server is shutdown, it will revert back to using the "default.xml" file when next started.

10.3 Adding a Second Server

To run a second server, change its name and input port number to something other than 4935, and give this server a different name. The other scripts and configuration files must be edited to accommodate the new name and ports.

10.4 Provided Configuration Files

The 'rt-stps/config' directory contains sample setup files. The sample XML files are as follows:

- a) default.xml – default configuration loaded by RT-STPS if a configuration file is not otherwise specified.
- b) aqua.xml – reads Aqua spacecraft telemetry from socket 4935 and writes PDS (packet file and Construction Record [CSR] file pairs) data to files and a port for each of the major instruments aboard the spacecraft.
- c) terra.xml – reads Terra spacecraft telemetry from socket 4935 and writes PDS and CSR metadata to files and a port for the MODIS instrument.
- d) npp.xml – reads Suomi NPP spacecraft telemetry and will produce RDRs for the ATMS, CrIS, and VIIRS instruments, with or without attitude and ephemeris (depending upon their presence in the data).
- e) npp_with_omps.xml – produces the same RDR output as npp.xml, with the addition of OMPS RDRs, allowing VIIRS, ATMS, CrIS, and OMPS RDRs to be produced simultaneously.

11 Alternate Server Interface

There is an additional command interface to the server accessible by software. This interface is useful to integrate the RT-STPS into a larger system with customization (e.g., connect it to scheduling software that may be modified to run RT-STPS).

The interface is through port 5935. (The port number is fixed.) The server expects to receive certain text string messages on this port for control. Each message should have the normal line terminator, and case is significant. The messages are commands to load and shut down sessions. The available commands are:

- a) loadgo <configurationFileName> - the server will load the configuration file from its configuration directory, and then enables itself for processing (e.g., "loadgo npp.xml");
- b) shutdown - stops processing and unloads the current configuration, which closes all output files;
- c) quit (or a "" string) - this terminates the connection through port 5935 and shuts down the server processing; the server then waits for a new connection.

The server does not send responses to any of these commands. The only feedback is via the server's console window. It will print the usual load and shutdown messages. It will also print error messages labeled as "ProxyThread" messages if it encounters them.

12 Understanding the Configuration Files

An RT-STPS XML configuration file defines the processing chain, virtual channels and packets of interest, and outputs. Its syntax is defined by the XML Document Type Definition (DTD) "rt-stps.dtd".

Each RT-STPS configuration file begins and ends with the rt-stps element and id such as:

```
<rt_stps id="NPP">
  <!-- configuration info here -->
</rt_stps>
```

The id is often the mission name. The id is not required. The id will be displayed by the viewer on the status line when it loads the configuration.

A detailed explanation of every configuration file element and attribute follows; along with examples for common usage.

12.1 Frame Synchronizer Element

Exactly one frame synchronizer (frame_sync) element may appear in a file. The element controls frame synchronization functions.

The frame_sync element determines if Pseudo-noise (PN) is removed from frames. Note that if processing Terra, the PN decoding happens in a different order and this is set by the specifying terra_decoder element (refer to Table 1).

The following is an example of a typical configuration:

```
<frame_sync frameLength="1024" PnEncoded="true">
  <timestamp epoch="19580101000000"/>
</frame_sync>
```

Table 1. frame_sync Element

Field	Default	Description
pattern	0x1ACFFC1D	The frame synchronization pattern. The default is the CCSDS standard. The number of characters in the pattern determines the pattern length. It must be at least two bytes long.
frameLength	1024	The frame length in bytes. If the frames contain Reed-Solomon parity, then specify a frame length that satisfies the Reed-Solomon decoder. For example, 1024 is the required frame length for RS interleave 4 frames. Or for interleave 5 frames, use 1264. Interleave 1 frames are 256 bytes. Please note that the frame lengths must include the 4-byte attached synchronization marker.
slip	0	In bits, 0 is typical. The field must have a value of 0, 1, or 2.
trueSync	true	True: searches for the pattern exactly as specified. If invertedSync is true, set this to false.
invertedSync	false	True: searches for an inverted sync pattern. Set trueSync to false if this field is true.
correctPolarity	true	If true, correct the polarity of frames that it determines have an inverted sync pattern. It inverts the entire frame and not just the pattern. This field is ignored unless invertedSync is true.
flywheelDuration	0	If non-zero, it will skip over this many blocks of "length" bytes before it again searches for the pattern. Zero is typical.
sendFlywheels	false	True: send on flywheel frames as if they were lock frames. False: discard them.
pnEncoded	false	True: assumes the frames are encoded with bit transition density encoding, and it will decode the frames. (Note: the PN decoder is specified as a separate link with the name "pn". It is embedded in the frame synchronizer setup because it does not have any additional setup fields).

Field	Default	Description
epoch	19950810000000 (Aug 10, 1995 00:00:00)	The epoch, sessionStart, and stepSize fields configure the clock used to create frame annotation. This may be important if annotation is being created with each packet, frame, or unit, or the output is EOS PDS files through the "sorcerer" element. The epoch sets the start time from which all times are measured. Its format is a string of the form: "YYYYMMDDhhmmss."
sessionStart	The current wall clock time.	This will be the time of the first data unit. By default, sets the session start time to the computer's current time. By changing session start, the session appears to run at a different date and time. The format for specifying sessionStart is the same as epoch. Omit this field to get the default behavior.
stepSize	0	Normally, the time difference between frames will be real time, after adjusting for epoch and the session start time. If this field is set to a positive value, then the annotation timestamps of successive frames will differ by this step size (in milliseconds), and the wall clock is ignored. For example, if stepSize is set to 100, then each frame's time will differ from the preceding one by 100 milliseconds. (Note: The frame synchronizer will not adjust the time to account for sync dropouts, so do not rely on the step size to detect lost frames. It will adjust for flywheel frames, dropped or not.)

12.2 Cyclic Redundancy Check (CRC) Decoder Element

Only one Cyclic Redundancy Check (CRC) decoder element may appear in a configuration. When set it checks frames for CRC errors and may discard frames if it detects errors. The frames must have 16-bit CRC parity. Otherwise, either omit this elements setup, or bypass it in the links setup.

Table 2. crc Element

Field	Default	Description
includeSyncPattern	false	When true, the CRC decoder will include the synchronization pattern in the CRC calculation. This is an atypical setup.
discardBadFrames	true	After the decoder calculates CRC and compares it against the parity in the frame, it can either discard the frame or pass it on to the next processing element. If it passes it on, a field in the quality annotation will mark it as a frame with a CRC error. (Note: The CRC error will appear as a count in the viewer's display. If frames with CRC errors are passed, be aware that subsequent elements may also encounter non-fatal processing errors depending on where in the frame the error occurred. In addition data units may be routed to incorrect destinations, or data units may have incorrect science or engineering data, because of the indeterminate bit errors in the frame).
offsetToParity	0	This is the byte offset from the frame start to the first byte of CRC parity, which is two bytes wide. In general, the CRC parity follows the frame data but precedes any Reed-Solomon parity. If this field is set to zero, then it calculates the value. Change it only if there is a non-standard location for the parity.
startSeed	0xFFFF	This is the start value for calculating the CRC parity. It is usually all ones or all zeroes. (Note: if there are CRC errors on every frame and there is certainty that the frames do contain CRC parity, try setting this field to zero.)

12.3 Reed-Solomon Decoder Element

Only one Reed-Solomon decoder element (`reed_solomon`) may appear in a configuration. It checks frames for block Reed-Solomon errors if they are Reed-Solomon encoded, and attempts to correct the errors if so configured. It may discard frames if it detects errors. It does not perform CCSDS VCDU header error detection and correction. If Reed-Solomon parity is present, then the length of frames is preset to certain absolute values. For interleaves 1 through 5, the standard CCSDS frame lengths are 256, 512, 760, 1024, and 1264 respectively. From a processing standpoint Reed-Solomon detection and correction is resource-intensive. If there are performance problems in a real-time environment, try turning off block correction.

An example in common usage for several NASA missions is as follows:

```
<reed_solomon useStandardCCSDS="true" doBlockCorrection="true"
discardUncorrectables="true" interleave="4"/>
```

Table 3. reed_solomon Element

Field	Default	Description
<code>interleave</code>	4	This field's value must match the spacecraft's Reed-Solomon interleave value. Only values between 1 and 5 inclusive are allowed.
<code>doBlockCorrection</code>	true	True: attempt to correct any frame errors. It will then mark the frame's quality annotation as corrected if it corrected the frame, or as uncorrectable if it could not fix the errors or if this field was set to false.
<code>discardUncorrectables</code>	true	True: discard any frame that it cannot correct, or any frame that has an error and <code>doBlockCorrection</code> was turned off. (Note: As with passing on frames with CRC errors, passing on frames with RS errors may cause errors in subsequent processing, as well as incorrect routing or corrupted science or engineering data).
<code>useStandardCCSDS</code>	true	True: use a standard CCSDS setup, which includes the <code>virtualFill</code> and <code>dual</code> fields as well as several hidden fields. (Note: This field should always be set to true.)
<code>virtualFill</code>	--	The number of virtual fill bytes in each frame.
<code>dual</code>	true	Dual mode or non-dual mode.

12.4 Spacecrafts

The spacecrafts element contains one or more spacecraft elements; only one spacecrafts element is allowed. Each child spacecraft element defines information about a different spacecraft. Typically only one is set for most configurations. A typical spacecraft definition is as follows:

```
<spacecrafts>
  <spacecraft label="spid1" id="157"/>
</spacecrafts>
```

Table 4. spacecrafts Element

Field	Default	Description
label	None	The label is a unique name for this element, and other items will refer to this element by the label. Provide a label that is unique to the entire configuration. Often it will set to the spacecraft name.
id	None	This is the spacecraft ID, a number is required.
insertZoneLength	0	Some CCSDS setups will have an Insert Zone embedded in every CADU. If present provide the zone length in bytes even if it will not be used later. The default is zero bytes, no Insert Zone present.
headerErrorControlPresent	false	If true, then each CADU VCDU header has special parity included in the CADU. If the error control field is present, set this field to true even if detecting or correcting VCDU header errors is needed.
doHeaderDecode	false	Not currently implemented. It may in the future be used to detect and correct errors in the CADU VCDU header. Header error control information and parity must be present if this field is true.

12.5 Terra Decoder Element

The Terra Decoder is a PN decoder for the Terra spacecraft that decodes a specific non-standard PN encoding. Omit it for any other spacecraft. The link name is its label. It has no arguments other than a required unique label field. It is typically specified as follows:

```
<terra_decoder label="TerraDecoder" />
```

12.6 CADU Service Element

Only one `cadu_service` element may appear in a configuration, and it defines the entry point for all CCSDS processing. The service routes the frames to different CCSDS services based upon virtual channel and spacecraft numbers. It is a list of one or more mappings of spacecraft ID and virtual channel ID pairs to CCSDS service labels.

The element name for a member of its map list is "svlink," and Table 5 describes "svlink" fields. Note that multiple (spid, vcid) pairs may map to the same target, and one (spid, vcid) pair may map to multiple targets. Frame CADUs with unmapped (spid, vcid) pairs are counted and discarded. Typically, it will map pairs to elements in the "ccsds_services" list, but this is not required. For example, it could map an "svlink" to an output channel. A short example of two virtual channels is as follows:

```
<cadu_service>
  <svlink vcid="1" spid="157" label="VCID1"/>
  <svlink vcid="6" spid="157" label="VCID6"/>
</cadu_service>
```

Table 5. cadu_service Element

Field	Default	Description
spid	None	A spacecraft ID, a number, which will be found inside a CADU. Required.
vcid	None	A virtual channel ID, a number, which will be found inside a CADU. Required.
label	None	The label of used in an element that expects CADUs or frames. All CADUs with matching spid and vcid will be sent to this item. Required.

12.7 CCSDS Services Element

The `ccsds_services` element lists CCSDS service options: `vcd`, `bitstream`, and `path`. They may be arranged in any order. Typically, the `cadu_service` element will link to these services.

12.7.1 VCDU Service Element

There may be more than one `vcd` element. Each one must have a unique label. The VCDU processing removes the VCDU from a CADU and sends it for further processing. It can also send Coded VCDUs (CVCDU), which are VCDUs with Reed-Solomon parity still attached. One VCDU node usually processes VCDUs for one virtual channel.

Table 6. vcd� Element

Field	Default	Description
label	None	The label uniquely identifies this item, and it must be different from any other item label in the configuration. The name typically incorporates the virtual channel number. Required.
spacecraft	None	A reference to a spacecraft label, which is defined in the spacecrafts list. Required.
discardRsParity	false	If true, the VCDU node will discard Reed-Solomon parity from each VCDU before forwarding it. The processing element gets the parity length from the Reed-Solomon node, so configure the Reed-Solomon decoder node to use this option. The Reed-Solomon element must be in the configuration file, but it need not be a linked element.

12.7.2 Bitstream Service Element

There may be more than one bitstream element. Each must have a unique label. The Bitstream node removes the BPDU from a CADU and sends it on to a node that accepts units. One Bitstream node usually processes BPDUs for one virtual channel. It does not merge BPDUs.

This processing element accounts for Reed-Solomon and CRC parity by searching for the existence of those decoder nodes, even if they are not linked into the pipeline. If the CADUs do not have Reed-Solomon parity, make sure to remove the Reed-Solomon element from the setup. Merely bypassing it is insufficient. Otherwise, the BPDUs will be truncated. The same is true for CRC parity; remove the element definition if CRC parity is not present.

Table 7. bitstream Element

Field	Default	Description
label	None	The label uniquely identifies this item, and it must be different from any other node label in the configuration. The name typically incorporates the virtual channel number. Required.
spacecraft	None	A reference to a spacecraft label, defined in the spacecrafts list. Required.
OCFpresent	false	If true, every CADU will contain a 32-bit Operational Control Field (OCF). The OCF (also known as the Command Link Control Word or CLCW) is echo information from the forward command link. The bitstream element uses this flag to compute the BPDU length. Incorrectly setting this field will cause BPDUs to be short or long by four bytes.
crcParityPresent	false	Bitstream processing uses this flag to compute the BPDU length. If true, it will subtract two bytes from the BPDU length. If false, it will look for the CRC Decoder node and flip this flag to true if it finds it, and then it will subtract two bytes. If this field is false, it will automatically detect for CRC parity presence. If true, it will assume CRC parity is present regardless of whether or not the CRC Decoder element is available.

12.7.3 Path Service Element

There may be more than one path element. Each one must have a unique label. The path service performs packet reassembly on a CADU's data zone and sends the packets on for further processing. One Path service usually processes CADUs from one virtual channel.

This service accounts for Reed-Solomon and CRC parity by searching for their existence even if they are not linked into the pipeline. If the CADUs do not have Reed-Solomon parity, make sure to remove the Reed-Solomon element from the setup. Merely bypassing it is insufficient. Otherwise, packets may be truncated, and the node will report numerous dropouts and sequence errors. The same is true for CRC parity; remove the element definition if CRC parity is not present.

Table 8. path Element

Field	Default	Description
label	None	The label uniquely identifies this node, and it must be different from any other node label in the setup. The name typically incorporates the virtual channel number. Required.
spacecraft	None	A reference to a spacecraft label, which is defined in the spacecrafts list. Required.
maxRationalPacketSize	8192	This is the maximum rational packet size in bytes. Make sure this field is larger than the packet size.
fill	0xC9	When the Path Service is unable to fill a packet in its entirety, it will fill the remainder by repeatedly appending this fill byte. (Note: it will discard any packet that does not have a packet header and at least one byte of real data.) It marks the packet annotation for packets with fill data.
OCFpresent	false	If true, the node expects that every CADU will contain a 32-bit Operational Control Field (OCF). The OCF (also known as the Command Link Control Word or CLCW) is echo information from the forward command link. This flag is used to compute the data zone length. Incorrectly setting this field will cause short packets and sequence errors because it will include OCF data in the data zone.
crcParityPresent	false	The Path processing uses this flag to compute the data zone length. If true, it will subtract two bytes from the data zone length. If false, it will look for the CRC Decoder element and flip this flag to true if it finds it, and then it will subtract two bytes. If this field is false, it will automatically check for CRC parity presence. If true, it will assume CRC parity is present regardless of whether or not the CRC Decoder is available.

Field	Default	Description
discardPacketsWithFill	false	If true, discard short packets to which it added fill data. If true, it will send them on, but it will also mark them in the packet annotation.
discardIdlePackets	true	If true, discard idle packets. Idle packets are fill packets that usually contain no useful data. (Note: Idle packets should usually be discarded unless there is other information in them that is needed. If it does send them on, it marks them as idle in the packet annotation.)

Packets are sorted by their application IDs, they may be forwarded to more than one recipient. The definition contains a list of one or more mappings of application IDs to packet processing labels using the pmlink element. The pmlink fields are described in Table 9. Packets with unmapped application IDs are counted and discarded. Typically, application IDs are mapped to elements in the packets list, but this is not required. For example, it is legal to map a pmlink to a packet output channel.

Table 9. pmlink Element

Field	Default	Description
appid	None	The application ID. Required.
label	None	The label of an item that accepts packets. Required.

An example for several packets from VCID1 is as follows:

```
<ccsds_services>
<path label="VCID1" spacecraft="spid1" maxRationalPacketSize="65542">
  <pmlink appid="101" label="FILL_101"/>
  <pmlink appid="515" label="ATMS_515"/>
  <pmlink appid="528" label="ATMS_528"/>
  <pmlink appid="530" label="ATMS_530"/>
  <pmlink appid="531" label="ATMS_531"/>
</path>
</ccsds_services>
```

12.8 Packets Element

The packets element defines a packet for each application ID. Its primary functions are to verify that each packet has an acceptable size and to look for sequence errors, which indicate a loss of one or more packets. Packet elements should correspond to pmlink definitions.

Several of the packet definitions that correspond to the packets element are as follows:

```
<packets>
  <packet appid="514" label="ATMS_514" minSize="7" maxSize="65542"/>
  <packet appid="515" label="ATMS_515" minSize="7" maxSize="65542"/>
  <packet appid="516" label="ATMS_516" minSize="7" maxSize="65542"/>
  <packet appid="517" label="ATMS_517" minSize="7" maxSize="65542"/>
</packets>
```

Table 10. packets Element

Field	Default	Description
label	None	The label uniquely identifies this item, and it must be different from any other node label in the setup. The name typically incorporates the application ID. Required.
appid	None	An application ID associated with this item. Optional, use for self documentation.
minSize	15	The minimum packet size that this node accepts. If it detects a packet with a length that is not within minSize and maxSize inclusive, it will mark the packet's annotation, and it may delete the packet if so configured.
maxSize	8192	The maximum packet size that this node accepts. If packets are one size, set maxSize and minSize to the same value.
discardWrongLengthPackets	true	If true, discards packets with sizes that are not within minSize and maxSize. Otherwise, it marks the packet annotation and sends them on to the next node.
checkSequenceCounter	true	If true, the node checks the packet sequence counters for packet gaps. It reports the number of gaps and cumulative count of missing packets in its status. It also marks packet annotation for packets that are near gaps.

12.9 Output Channels Element

The `output_channels` element defines a list of output channels. Any number may be defined; one may accept information from multiple sources. Most output channel elements will accept only one data unit type: packets, frames, or units. A number of forms of outputs are supported, including: TCP/IP sockets, files, PDS files, RDR files and null. (The null channel simply discards data. Its only field is its label, which must be unique.) For most applications, the output channel area will accept packets that are listed in the links element area as described below.

12.9.1 File Output Channel Element

The element name is "file," and there may be more than one element.

Table 11. file Element

Field	Default	Description
label	None	The label uniquely identifies this item, and it must be different from any other label in the setup. Required.
unitType	None	Enumerated values are: UNIT, PACKET, or FRAME. This field describes the type of data unit, required.
annotation	None	Enumerated values are: NONE, BEFORE, or AFTER. This field determines if annotation is to be written with each data unit and where it should go. Frames and units get frame annotation. Packets get packet annotation followed by frame annotation. NONE means the data units are written without annotation. Use BEFORE to write the annotation before each data unit. Use AFTER to append the annotation.
directory	--	The output file directory. If omitted files will be created in the default data directory, which is usually 'data'. The default directory is defined as an argument in the script that starts the server. If using a different directory, specify it here.
filename	--	The name of the file that will be created. If <code>autoGenerateFilename</code> to true, then omit this field.
userLabel	--	This is an optional label that the node inserts into the file name when <code>autoGenerateFilename</code> to true.

Field	Default	Description
autoGenerateFilename	true	If true, the node generates a unique filename that should not overwrite an existing file. The file name will have the form "t"+ "p or f or g" + "yyyyDDDHHmmss"+ "userLabel" + ".dat". The "t" stands for "telemetry." It then inserts "p", "f", or "g" for "packets", "frames", or "general" units. It finally adds the date and a user label.

12.9.2 Annotation

Annotation refers to quality and time, and it may be optionally appended or prefixed to each data unit. The sorcerer output channel creates EOS Terra or Aqua spacecraft Production Data Set (PDS) and Expedited Data Set (EDS) files. PDS and EDS files each include a Construction Record (CSR) and one or more packet files containing un-annotated packets. If an annotation option is chosen, then RT-STPS writes 64 bits of frame annotation with each frame, packet, or unit. In the packet case, it also writes 32 bits of packet annotation before the frame annotation.

Table 12. Packet Annotation

Bits	Packet Annotation (32 bits) Precedes Frame Annotation
31-18	Not used (most significant bit)
17	1= packet has invalid length, which is outside the configured minimum and maximum packet length for this packet stream.
16	1= this packet could not be constructed in its entirety, and so it has appended fill data.
15-0	The number of "good" bytes in this packet. For complete packets, it is the packet length. For packets with fill, it is the index of the first fill byte.

Use the following to create a plain file output channel:

```
<file autoGenerateFilename="true" label="ENG_OUT_FILE" unitType="PACKET"
directory="./data" annotation="NONE" userLabel="ENG_OUT"/>
```

Table 13. Frame Annotation

Bits	Frame Annotation (2 x 32 bits)
31-26	Not used (most significant bit)
25	1= Frame contains an idle/fill VCDU (CCSDS state).
24	1= Frame has bad first header pointer (CCSDS error).
23	1= Path Service had problem composing a packet from this frame.
22	1= sequence error between this frame and preceding frame
21	1= Frame is Reed-Solomon uncorrectable.

20	1= Frame is Reed-Solomon corrected.
19	1= Frame has CRC error.
18	1= Slipped frame (Frame was aligned.)
17	1= Inverted frame (Polarity was corrected.)
16	1= Lock frame
15-0	Day of year (1-366)
31-0	Milliseconds of day

12.9.3 Socket Output Channel Element

To use socket outputs, there must be a client at the target computer waiting at a server socket to establish a connection. If there is no server, then trying to load a configuration with socket output will fail.

A socket connection sending data units to a server can be a considerable bottleneck for RT-STPS because there is a single control loop through the entire processing chain including the output channels, essentially meaning that processing will run no faster than the slowest processing element, including output channels. If this problem occurs, consider using the Rate Buffering Program (Rat), described in section 9.6.

The following example shows a socket definition which will feed another remote application.

```
<output_channels>
<socket label="modis_socket" unitType="PACKET" annotation="NONE"
host="localhost" port="3511" bufferSize="65536" />

</output_channels>
```

Table 14. socket Element

Field	Default	Description
label	None	The label uniquely identifies this item, and it must be different from any other node label in the configuration. Required.
unitType	None	Enumerated values: UNIT, PACKET, or FRAME. This field describes the type of data unit accepted by this item. Required.
annotation	None	Enumerated values: NONE, BEFORE, or AFTER. This field determines if annotation is to be written with each data unit and where it should go. Frames and units get frame annotation. Packets get packet annotation followed by frame annotation. NONE means the data units are written without annotation. BEFORE means the annotation is written before each data unit. AFTER means annotation is appended.
bufferSize	8192	The node configures the output socket to use this buffer size. To improve network performance, experiment with this number.
host	None	The host name or IP address of the target computer. Required.
port	None	The port number of the target computer for the target connection.

12.9.3.1 Sorcerer Output Channel Element

Define the sorcerer element to create one or more EOS PDS or EDS file sets. A file set consists of a Construction Record (CSR) file and packet files. A data file contains packets without annotation for up to three application IDs. Consult an EOS Interface Control Document (ICD) to understand the significance of some of these fields.

Table 15. sorcerer Element

Field	Default	Description
label	None	The label uniquely identifies this item, and it must be different from any other node label in the setup. Required.
major	0	The major version number. This value is inserted into the construction record and has no effect on processing.
minor	0	The minor version number. This value is inserted into the construction record and has no effect on processing.
spid	42	The spacecraft ID. This value is inserted into the construction record and has no effect on processing.
path	--	The output file directory. If omitted files will be created in the default data directory, which is usually "data". The default directory is defined as an argument in the script that starts the server. Choose to put your files in a different directory by specifying it here.
datasetCounter	0	A number that is embedded in the construction record and the file name.
create	--	The creation date for the output files, which is inserted into the construction record. If omitted, the current date and time will be used. If provided, the format is: "yyDDDHHmmss".
KBperFile	0	Kilobytes per file. If set to zero, the construction record file and data file will contain all packets from the session. If set to a positive number, it will split the data file into a set of data files, approximately this size. Each file name will have a sequence number embedded in it. Note that the file name field has room for only 99 data files.

Field	Default	Description
test	false	A flag embedded in the construction record to indicate that the PDS contains test data instead of real data.
type	PDS	Append "PDS" or "EDS" file to the file. It does not affect processing except when QuicklookEDS is set to "true".
quicklookEDS	false	This field is ignored unless the type is "EDS". When true and type is "EDS," only writes packets that have the quicklook flag enabled in their secondary header. Do not turn this flag on for non-Terra application IDs that do not use the Terra secondary header format.
discardBadLengthPackets	True	If true, discards packets with an incorrect length. Otherwise, it will write them.

12.9.3.2 Application ID Sub-element

Each appid sub-element sets up one application ID. There must be at least one appid element, but there may be no more than three. Each appid may also contain a list of valid packet lengths. Specify a range of valid lengths, or list the lengths individually.

Table 16. appidSub-element

Field	Default	Description
id	None	Specifies the application ID number. This field is required.
vcid	None	The virtual channel number from which this application ID came. An application ID may come from a maximum of two virtual channels. Specify the second one in vcid2. This field is required.
vcid2	None	A second virtual channel number. Omit this field if there is no second virtual channel for this application ID.
spid	42	The spacecraft ID. (e.g., 154 is the ID for Aqua.)
CUCtime	false	This field determines the expected packet secondary header format. If it is not set correctly, then the time fields in the construction record will have an incorrect format. Set this to false for all Terra application IDs. For Aqua, consult the ICD. Some applications IDs use a nine byte secondary header (all Terra) in which the time field is in CCSDS Day Segmented format.
stepsize	1	The packet sequence number step size. Do not change this number. Sorcerer uses it to determine the gap size for missing packets.

Field	Default	Description
minLength	--	The minimum packet length. It must be at least 15 and not bigger than maxLength. There are two ways to configure packet lengths. Either set minLength and maxLength, or provide a list of "packetLength" elements, one for each length. If provided in a list, minLength and maxLength are ignored. However, one of the two methods must be used.
maxLength	--	The maximum packet length.

12.9.3.3 Packet Length Sub-element

The sub-element name is packetLength. Each statement defines a valid packet length for an application ID in this PDS. Some application IDs may have more than one packet length; list them here, one per statement. The following example is typical of sorcerer elements:

```
<output_channels>
<sorcerer label="GBAD" path="../data" spid="154">
<appid id="957"vcid="3"minLength="126" maxLength="126"
timeOffset="6" CUtime="true" />
</sorcerer>
<socket label="modis_socket" unitType="PACKET" annotation="NONE"
host="localhost" port="3511" bufferSize="65536" />

</output_channels>
```

Table 17. packetLengthSub-element

Field	Default	Description
length	None	A packet length, required.

12.9.4 RDR Output Element

The RDR output element defines an RDR output formatted file. One or more RDR outputs may be defined. RT-STPS supports the creation of RDRs for the Suomi NPP VIIRS, ATMS, CrIS, and OMPS instruments. Attitude and ephemeris packets included in the input produce the corresponding Spacecraft Diary with each RDR.

Table 18. RDR Element

Field	Default	Description
label	None	The label uniquely identifies this node, and it must be different from any other node label in the setup, required.
directory	--	The output file directory. Required. The RDR will be created in the specified location if it exists.
mission	Suomi NPP	Only Suomi NPP-compatible RDRs are supported at this time. This field is currently not checked by the internal software and is hardcoded to Suomi NPP. Reserved for future use.

12.9.4.1 Packet List

The RDR created is dependent on the packet list defined and linked to it. For example, in order to output a VIIRS RDR file, the appropriate VIIRS packets must first be defined in the packet tag area (along with related CCSDS services and path tags). These packets of interest must be defined in the links area that refers to the RDR of interest. If attitude and ephemeris are desired, their various services, path and packet tags must be defined, and then a link defined to them for the VIIRS RDR. The same process would be used to output RDRs for ATMS, CrIS, and OMPS.

The software automatically senses which RDR to build based on its input packets; the proper packet list must be defined so that the CCSDS packet processing portion of RT-STPS passes the correct packets to it. If this is not the case an RDR will not be created (i.e., if no application identifiers match the internal application identifier table for each RDR).

12.9.4.2 Supported Application Identifiers

RT-STPS processes a raw packet input file into an RDR. It will filter unwanted packets that do not constitute the particular RDR of interest with the proper configuration file. For example, if a VIIRS RDR is being created, the application identifiers for that RDR should be defined in the configuration file. Table 19 contains the supported application IDs by type.

Table 19. Supported Application Identifiers

Type	Application IDs
VIIRS	800 – 823, 825 and 826
ATMS	515, 528, 530, 531
CrIS	1315-1395, 1289 and 1290
OMPS	560-563
Spacecraft Diary	11

In order to create an RDR file with Spacecraft Diary, RT-STPS should be configured to

filter the science packets of interest, and it should include packets with appid 11 into the output file. The resulting configuration file may then be used to produce the RDRs of interest along with the Spacecraft Diary.

Table 20. Supported Application Identifiers by RDR

RDR	Application IDs
VIIRS and Spacecraft Diary	800 – 823, 825 and 826 and 11
ATMS and Spacecraft Diary	515, 528, 530, 531 and 11
CrIS and Spacecraft Diary	1315-1395, 1289, 1290 and 11
OMPS and Spacecraft Diary	560-563, and 11

12.10 Links Element

The links element links processing modules or outputs together using the "from" and "to" fields.

Table 21. links Element

Field	Default	Description
from	None	The source node from which data units are sent. This is an element label and is required.
to	None	The destination node to which data units are sent. This is an element label and is required.

Table 22 contains labels are predefined to configure the processing chain.

Table 22. Predefined Labels

Node	Label
Frame Synchronizer	frame_sync
PN Decoder	pn
CRC Decoder	crc
Reed-Solomon Decoder	reed_solomon
Frame Status	frame_status
CADU Service	cadu_service

A typical links path from the frame synchronizer to CADU Service is as follows:

```
<link from="frame_sync" to="pn" />
<link from="pn" to="crc" />
<link from="crc" to="reed_solomon" />
<link from="reed_solomon" to="frame_status" />
<link from="frame_status" to="cadu_service" />
```

Modify this links path as needed. For example if CRC decoding is not being used, then remove the CRC lines and substitute: link from="pn" to="reed_solomon". Then add output channel links to the list as in this example:

```
<link from="vcdu18" to="file1" />  
<link from="bitstream30" to="file2" />  
<link from="a256" to="file3" />  
<link from="ENG_11" to="ATMS_SCI_AE_RDR"/>  
<link from="ATMS_515" to="ATMS_SCI_AE_RDR"/>  
<link from="ATMS_528" to="ATMS_SCI_AE_RDR"/>  
<link from="ATMS_530" to="ATMS_SCI_AE_RDR"/>  
<link from="ATMS_531" to="ATMS_SCI_AE_RDR"/>
```

Other links to other outputs would also be defined here.